Cahier des charges techniques

Sommaire

1. Contexte du projet

1.1. Présentation du projet

Votre agence web a été sélectionnée par le comité d'organisation des jeux olympiques de Los Angeles 2028 pour développer une application web permettant aux organisateurs, aux médias et aux spectateurs de consulter des informations sur les sports, les calendriers des épreuves et les résultats des JO 2028.

Votre équipe et vous-même avez pour mission de proposer une solution qui répondra à la demande du client.

1.2. Date de rendu du projet

Le projet doit être rendu au plus tard le 7/11/2024.

2. Besoins fonctionnels

Le site web devra avoir une partie accessible au public et une partie privée permettant de gérer les données.

Les données seront stockées dans une base de données relationnelle pour faciliter la gestion et la mise à jour des informations. Ces données peuvent être gérées directement via le site web à travers un espace administrateur.

3. Ressources nécessaires à la réalisation du projet

3.1. Ressources matérielles

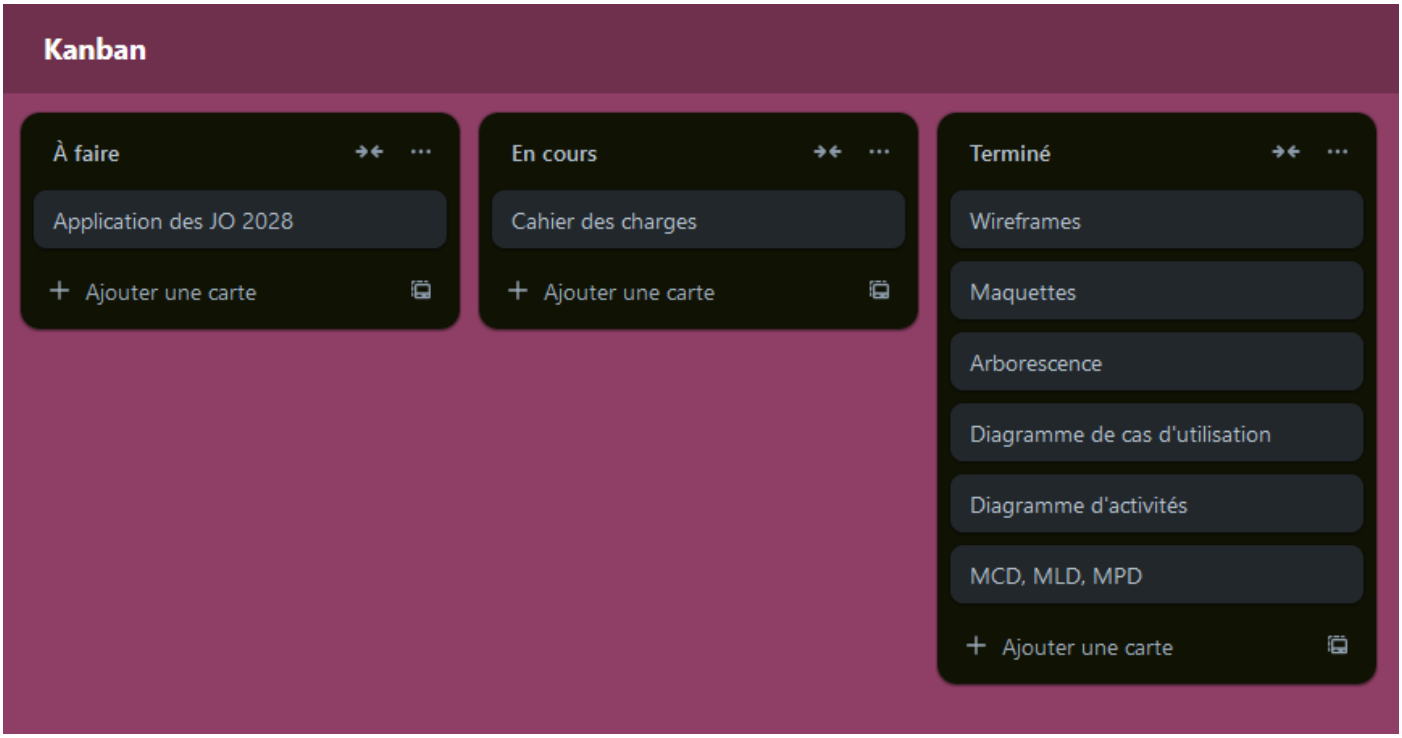
PC Fixe composé d'une unité centrale, d'un écran, d'une souris et d'un clavier.
L'accès à internet relié par Wi-Fi.

3.2. Ressources logicielles

IDE(Environnement de développement) : Visual Studio Code
Github (Hébergeur, dépôt et plateforme de développement collaboratif)
Apache (Serveur web contenu dans MAMP)
BDD relationnelle (Contenu dans MAMP et MySQL)
Outil de gestion de projet : Trello
UML : Visual Paradigm
Maquetage : Figma

4. Gestion du projet

Pour réaliser le projet, nous utiliserons la méthode Agile Kanban. Nous utiliserons également l'outil de gestion de projet en ligne Trello.

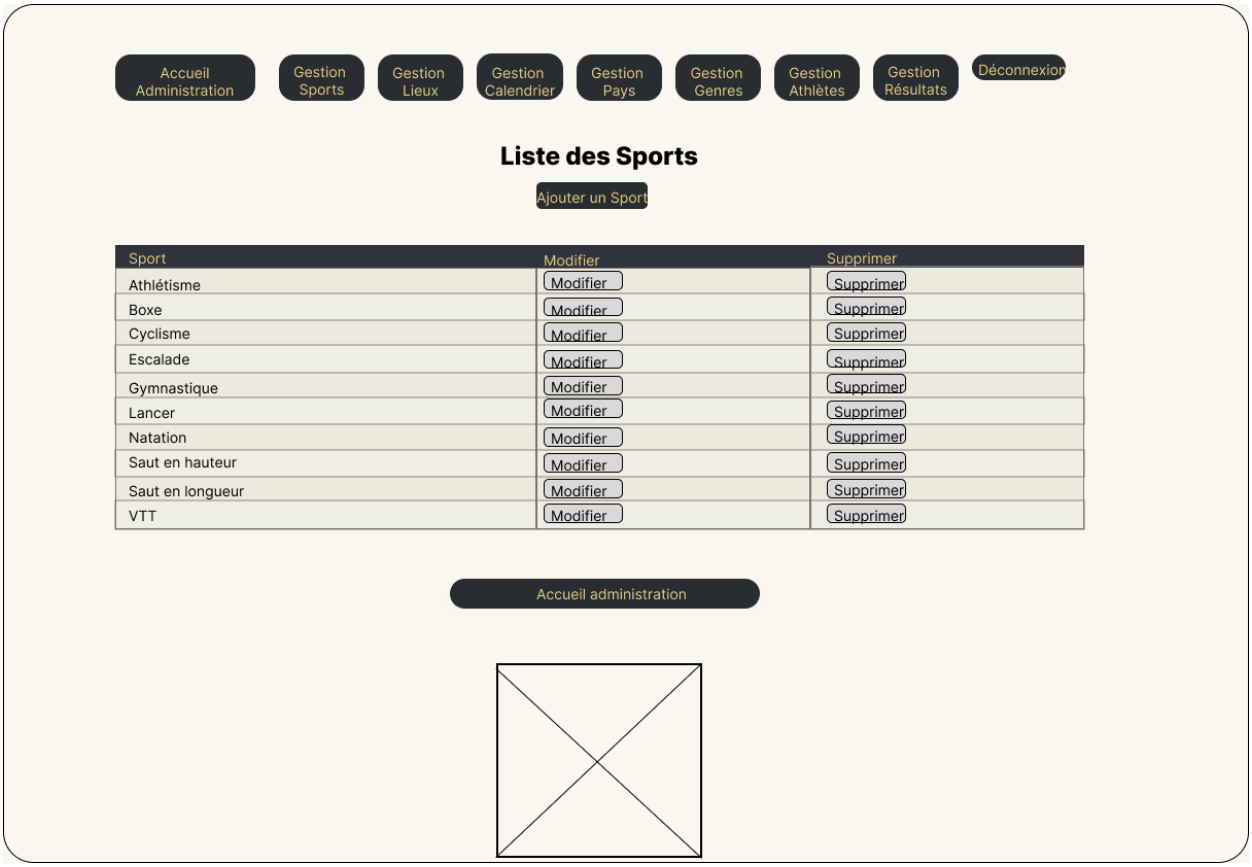


Nous travaillons également sur GitHub, plateforme de développement collaboratif.

5. Conception du projet

5.1. Le front-end

5.1.1. Wireframe

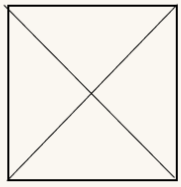


Accueil | Sports | Calendrier des épreuves | Résultats | Accès administrateur

Sports

Calendrier des épreuves

Résultats



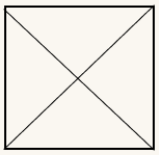
Accueil | Sports | Calendrier des évènements | Résultats | Accès administrateur

Connexion

Login :

Mot de passe :

Se connecter



5.1.2. Maquettes

Accueil | Sports | Calendrier des évènements | Résultats | Accès administrateur

Connexion

Login :

Mot de passe :

Se connecter

Accueil | Sports | Calendrier des épreuves | Résultats | Accès administrateur

Sports

Calendrier des épreuves

Résultats

Accueil Administration | Gestion Sports | Gestion Lieux | Gestion Calendrier | Gestion Pays | Gestion Genres | Gestion Athlètes | Gestion Résultats | Déconnexion

Liste des Sports

Ajouter un Sport

Sport	Modifier	Supprimer
Athlétisme	Modifier	Supprimer
Boxe	Modifier	Supprimer
Cyclisme	Modifier	Supprimer
Escalade	Modifier	Supprimer
Gymnastique	Modifier	Supprimer
Lancer	Modifier	Supprimer
Natation	Modifier	Supprimer
Saut en hauteur	Modifier	Supprimer
Saut en longueur	Modifier	Supprimer
VTT	Modifier	Supprimer

Accueil administration

Accueil

Sports

Calendrier des évènements

Résultats

Accès administrateur

Sports

Calendrier des épreuves

Résultats

Accueil

Sports

Calendrier des évènements

Résultats

Accès administrateur

Connexion

Login :

Mot de passe :

Accueil Administration

Gestion Sports

Gestion Lieux

Gestion Calendrier

Gestion Pays

Gestion Genres

Gestion Athlètes

Gestion Résultats

Déconnexion

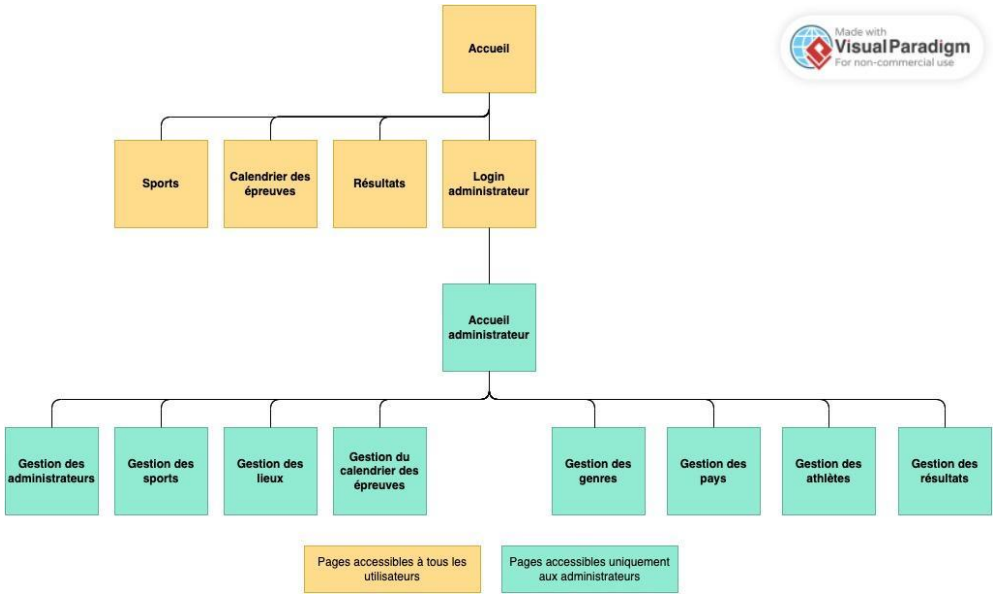
Liste des Sports

Ajouter un Sport

Sport	Modifier	Supprimer
Athlétisme	Modifier	Supprimer
Boxe	Modifier	Supprimer
Cyclisme	Modifier	Supprimer
Escalade	Modifier	Supprimer
Gymnastique	Modifier	Supprimer
Lancer	Modifier	Supprimer
Natation	Modifier	Supprimer
Saut en hauteur	Modifier	Supprimer
Saut en longueur	Modifier	Supprimer
VTT	Modifier	Supprimer

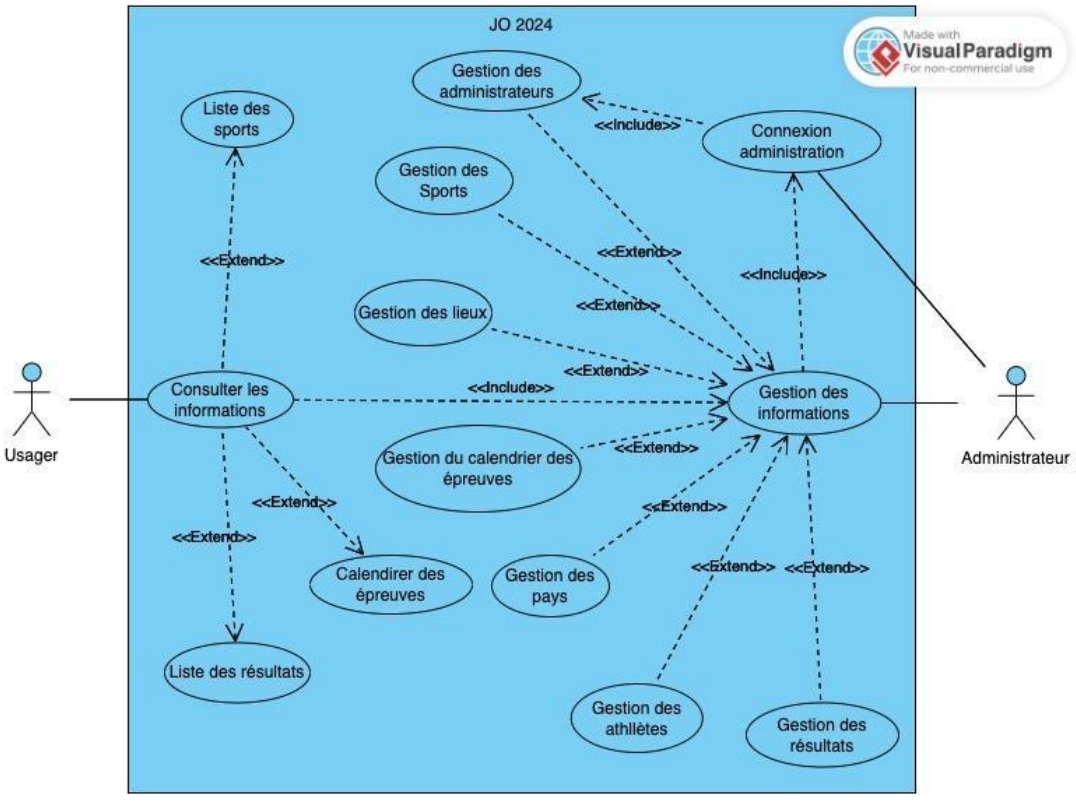
Accueil administration

5.1.3. Arborescences

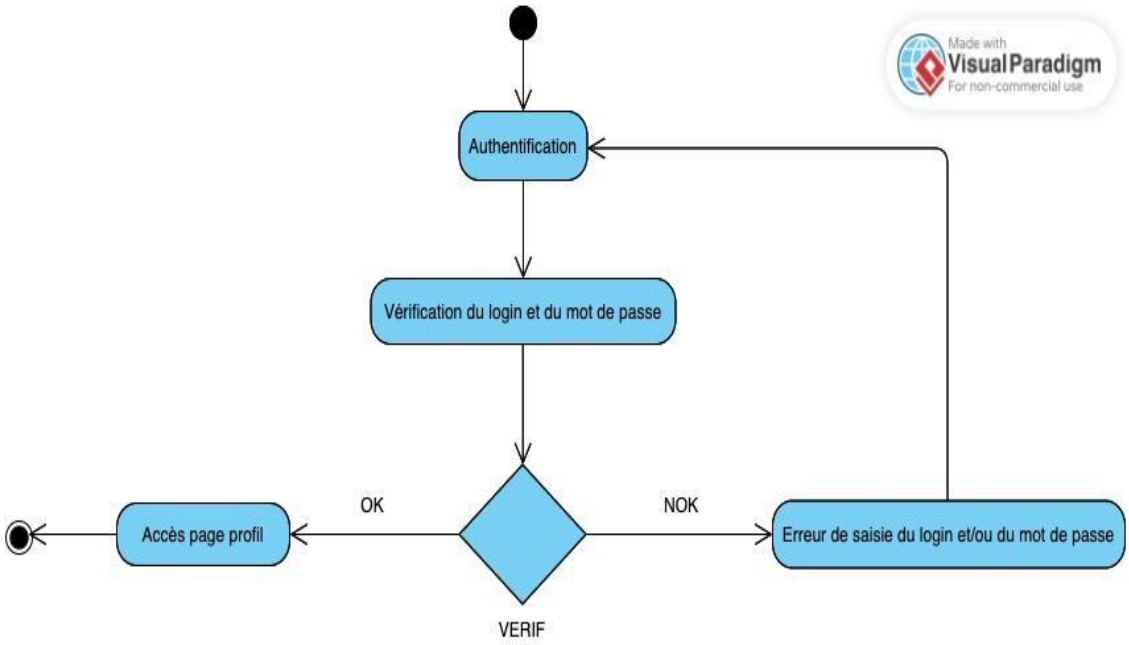


5.2. Le back-end

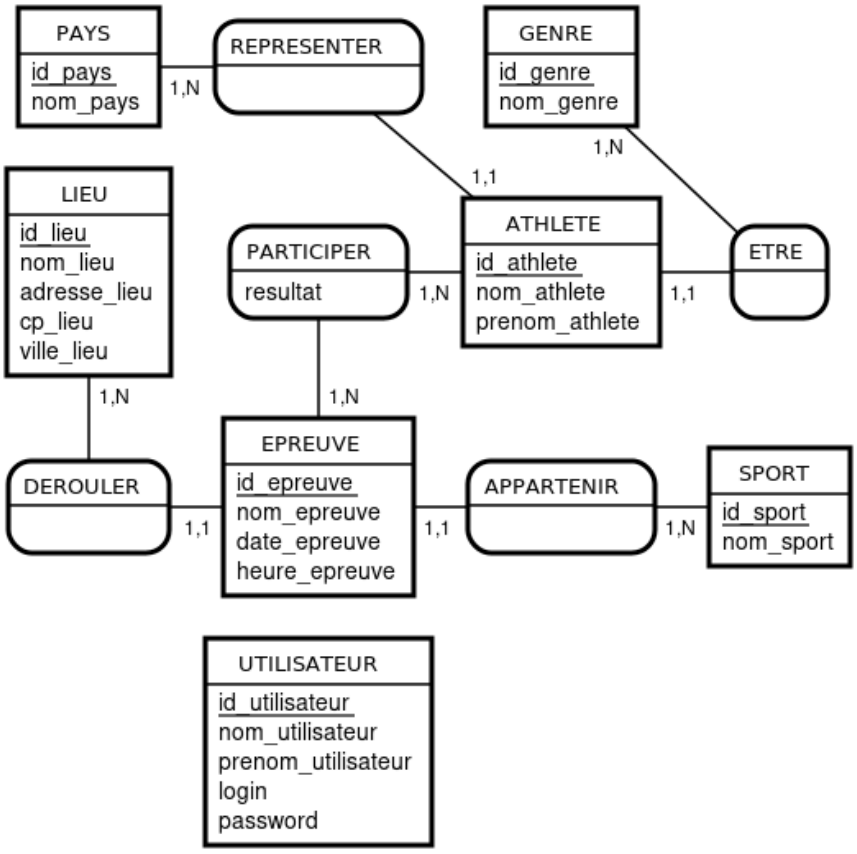
5.2.1. Diagramme de cas d'utilisation



5.2.2. Diagramme d'activités



5.2.3. Modèles Conceptuel de Données (MCD)



5.2.4. Modèle Logique de Données (MLD)

∉ ATHLETE (id_athlete, nom_athlete, prenom_athlete, id_pays, id_genre)
Clé primaire : id_athlete
Clés étrangères : id_pays en référence à id_pays de PAYS
id_genre en référence à id_genre de GENRE

- ∉ EPREUVE (id_epreuve, nom_epreuve, date_epreuve, heure_epreuve, id_lieu, id_sport)
  Clé primaire : id_epreuve
  Clés étrangères : id_lieu en référence à id_lieu de LIEU
id_sport en référence à id_sport de SPORT

- ∉ GENRE (id_genre, nom_genre)
  Clé primaire : id_genre
- ∉ LIEU (id_lieu, nom_lieu, adresse_lieu, cp_lieu, ville_lieu)
  Clé primaire : id_lieu

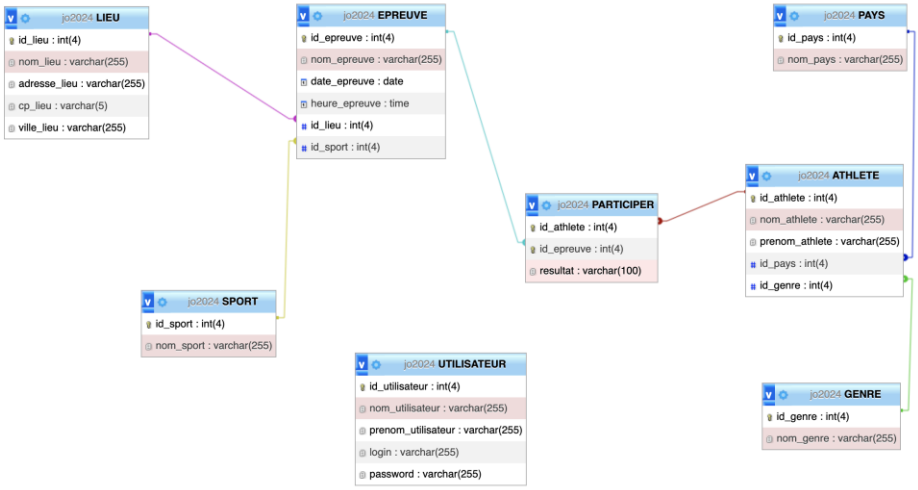
- ∉ PARTICIPER (id_athlete, id_epreuve, resultat)
  Clé primaire : id_athlete, id_epreuve
  Clés étrangères : id_athlete en référence à id_athlete de ATHLETE
id_epreuve en référence à id_epreuve de EPREUVE

- ∉ PAYS (id_pays, nom_pays)
  Clé primaire : id_pays, id_epreuve

- ∉ SPORT (id_sport, nom_sport)
  Clé primaire : id_sport

- ∉ UTILISATEUR (id_utilisateur, nom_utilisateur, prenom_utilisateur, login, password)
  Clé primaire : id_utilisateur, id_epreuve

5.2.5. Modèle Physique de Données (MPD)



6. Technologies utilisées

6.1. Langages de développement Web

HTML 5

CSS
Javascript (Vanilla)

6.2. Base de données

PHP
SQL

7. Sécurité

7.1. Login et protection des pages administrateurs

- **Formulaire de connexion :** L'utilisateur saisit son nom d'utilisateur et son mot de passe via une interface (par exemple, un formulaire sur une page web).
- **Vérification des identifiants :** Quand l'utilisateur soumet le formulaire, le serveur (backend) vérifie si les informations correspondent à celles enregistrées dans une base de données. Les mots de passe sont généralement **hachés** pour plus de sécurité. Si l'identifiant et le mot de passe correspondent, l'utilisateur est considéré comme authentifié.
- **Sessions :** Une fois l'utilisateur authentifié, une **session** est créée. Cela permet au serveur de se souvenir que cet utilisateur est connecté lors de ses prochaines requêtes. Ces informations sont stockées de manière sécurisée sur le serveur et reliées à l'utilisateur via un identifiant de session.
- **Protection des pages :** Pour les pages protégées (par exemple, un tableau de bord utilisateur), le serveur vérifie d'abord si une session valide est active (c'est-à-dire si l'utilisateur est connecté). Si ce n'est pas le cas, l'utilisateur est redirigé vers la page de connexion.
- **Déconnexion :** L'utilisateur peut se déconnecter, ce qui détruit la session en cours, supprimant ainsi toutes les informations relatives à l'utilisateur connecté. Cela empêche d'accéder aux pages protégées sans se reconnecter.

7.2. Cryptage des mots de passe avec Bcrypt

Pour sécuriser les mots de passe en PHP, **bcrypt** est utilisé avec les fonctions `password_hash()` et `password_verify()`.

1. **Hachage du mot de passe** : Lorsqu'un utilisateur s'inscrit, utilisez `password_hash()` pour créer un mot de passe sécurisé. Cela génère un hachage unique avec un **sel** aléatoire intégré :
- **Stockage** : Stockez ce hachage dans la base de données à la place du mot de passe en clair.
- **Vérification** : Lors de la connexion, comparez le mot de passe saisi avec le hachage en base en utilisant password_verify() :

En php :
$hashed_password = password_hash($plain_password, PASSWORD_DEFAULT);

7.3. Protection contre les attaques XSS (Cross-Site Scripting)

Le cross-site scripting est un type de faille de sécurité des sites web permettant d'injecter du contenu dans une page, provoquant ainsi des actions sur les navigateurs web visitant la page.

La protection contre les attaques **XSS (Cross-Site Scripting)** consiste à empêcher l'injection de scripts malveillants dans une application web. Voici quelques solutions courantes pour se protéger contre ces attaques :

1. **Échapper les entrées utilisateur** : Toujours échapper ou encoder les données utilisateur avant de les afficher dans des pages web. En PHP, utilisez `htmlspecialchars()` pour transformer les caractères spéciaux en entités HTML.
- ∉ **Validation des entrées** : Validez et filtrez les données saisies par l'utilisateur avant de les traiter. Utilisez des filtres comme `filter_input()` pour vous assurer que les données sont bien conformes au format attendu.
- ∉ **Utiliser des en-têtes HTTP de sécurité** :
- ∉ **Content Security Policy (CSP)** : Une politique CSP limite les sources de scripts exécutables sur vos pages, empêchant ainsi l'exécution de scripts malveillants.
- ∉ **X-XSS-Protection** : Activez cette option pour demander au navigateur de bloquer les scripts suspects.
- ∉ **Limiter l'injection de contenu** : Évitez d'accepter et d'afficher du code HTML brut provenant d'utilisateurs. Si vous devez afficher du contenu généré par l'utilisateur (comme des commentaires), limitez les balises HTML permises ou utilisez un éditeur qui nettoie le HTML, comme Markdown.
- ∉ **Utiliser des frameworks sécurisés** : Les frameworks modernes (comme Laravel, Django) incluent des mécanismes intégrés pour se protéger contre les attaques XSS en échappant automatiquement les données.

7.4. Protection contre les injections SQL

La protection contre les **injections SQL** vise à empêcher qu'un utilisateur malveillant n'exécute des requêtes SQL non autorisées via des entrées utilisateur. Voici les principales mesures de protection :

Utiliser des requêtes préparées (requêtes paramétrées) : Les requêtes préparées avec des **paramètres liés** empêchent l'injection directe dans les requêtes SQL. Cela permet au serveur de distinguer les données des commandes SQL. En PHP avec PDO :

```
$stmt = $pdo->prepare("SELECT * FROM utilisateurs WHERE username = :username");
$stmt->bindParam(':username', $username);
$stmt->execute();
```

Éviter d'insérer des données directement dans les requêtes SQL : Ne jamais insérer directement des données utilisateur dans une requête SQL en concaténant des chaînes, car cela ouvre la porte à des injections.

Validation et filtration des entrées : Validez et filtrez les données en fonction de leur type attendu (nombres, chaînes, etc.) avant de les utiliser dans une requête. Par exemple, utilisez `filter_var()` pour vérifier les adresses e-mail ou les entiers.

Limiter les privilèges des comptes de base de données : Assurez-vous que le compte utilisé pour se connecter à la base de données a uniquement les privilèges nécessaires (lecture/écriture).

N’utilisez jamais un compte administrateur pour des actions simples comme lire ou écrire des données utilisateur.

Utiliser ORM ou des frameworks sécurisés : Si possible, utilisez un **ORM (Object Relational Mapping)** ou un framework sécurisé (comme Eloquent dans Laravel) qui protège automatiquement contre les injections SQL.

Ces pratiques permettent de rendre votre application beaucoup plus résistante aux tentatives d'injection SQL.